# Praktikum µProzessor-Hardware

## Thema: GPS Daten Logger

## SS 2004

Andreas Schibilla (II4900)
Sebastian Reuss (II5062)

# 1. Inhaltsverzeichnis

## 2. Allgemeine Problemstellung

In diesem Praktikum soll ein Datenlogger für ein GPS Handgerät erstellt werden. Das GPS Handgerät der Firma Garmin hat einen internen Speicher in dem man eine abgelaufene Route (beispielsweise bei einer Wanderung) speichern kann. Allerdings ist dieser interne Speicher für längere Wanderungen nicht ausreichend groß. Deshalb soll ein Gerät entwickelt werden, dass die bereits gespeicherten Daten aus dem GPS Empfänger ausliest und in einem SRAM speichert. Später soll dieser Speicher dann auch wieder ausgelesen und die Daten an einen PC weitergeben werden können. Dafür soll der Mikrokontroller 89C2051 der Firma Atmel eingesetzt werden.

**Details**

Der AT89C2051 ist mit einer seriellen Schnittstelle ausgestattet. Diese soll für die Kommunikation mit dem GPS Empfänger und auch mit dem PC benutzt werden. Die Kommunikation soll auf 9600 bps,8 Datenbits und 1 Stopbit eingestellt sein. Die Kommunikation mit dem Garmin-GPS-Empfänger findet mit einem speziellen Protokoll statt. Der GPS Empfänger speichert unter anderem sogenannte Way- und Track-Points. Diese sollen beide ausgelesen werden. Das muss man nacheinander machen. Der Ablauf ist aber fast identisch. Wenn die Daten vom PC abgefragt werden, soll sich die Schaltung so verhalten, als ob sie selber ein Garmin GPS-Empfänger ist.

Eine Kompression der Daten ist wünschenswert, um möglichst viele Daten speichern zu können. Ebenfalls wichtig ist, dass die Schaltung auf die Anfrage des PCs korrekt reagiert. Wird eine Anfrage nach den Way-Points gestellt sollen auch nur diese zurück gesendet werden. Analog verhält es sich mit den Track-Points. Es sollte auch möglich sein, mehrere „Sessions“ aus dem GPS Empfänger auszulesen (falls die Wanderung mal wieder etwas länger ausfällt...).

Für die Speicherung steht ein SRAM der Größe 512k x 8 zur Verfügung. Allerdings benötigt man für die Adressierung von 512k bereits 19 Adressleitungen. Das sprengt die zur Verfügung stehende Pinanzahl des Atmels deutlich. Gefragt ist hier eine Lösung, die mit weniger Pins auskommt.

Weiteres Qualitätsmerkmal der Schaltung soll der Stromverbrauch sein. Da die Schaltung die Daten in einem SRAM speichert, muss die Schaltung natürlich permanent versorgt werden, beispielsweise aus einem Batteriepack. Hier soll darauf geachtet werden, dass die Praktikum µProzessor-Hardware-Schaltung so wenig Strom wie möglich verbraucht, wenn Sie eigentlich gerade nichts zu tun hat (Standby Modus).

**Abfrage von Track/Way Points aus dem GPS-Empfänger**

Die folgenden Paketbezeichnungen werden in dem Dokument: "Garmin GPS Interface Specifikation" erläutert.

| Sender | Paketbezeichnung | Packet-ID | Kommentar |
|---|---|---|---|
| PC | Pid_Product_Rqst | 254 | |
| GPS | Pid_Ack_Byte | 6 | |
| | Pid_Product_Data | 255 | |
| PC | Pid_Ack_Byte | 6 | |
| GPS | Pid_Protocol_Array | 253 | |
| PC | Pid_Ack_Byte | 6 | |
| -- | -- | | Bis hier identisch für Way- und Track-Points |
| PC | Pid_Command_Data | 10 | Hier: Cmnd_Transfer_Trk für Abfrage der Track-Points, oder Cmnd_Transfer_Wpt für Abfrage der Way-Points |
| GPS | Pid_Ack_Byte | 6 | |
| | Pid_Records | 27 | |
| PC | Pid_Ack_Byte | 6 | |
| GPS | Pid_Trk_Hdr | 99 | !!! Achtung: Nur bei Abfrage von TrackPoints !!! |
| PC | Pid_Ack_Byte | 6 | !!! Achtung: Nur bei Abfrage von TrackPoints !!! |
| GPS | Pid_Trk_Data oder<br>Pid_Wpt_Data | 34<br>35 | Schleife von Data und Ack, bis alle Daten gesendet wurden |
| PC | Pid_Ack_Byte | 6 | dito |
| GPS | Pid_Xfer_Cmplt | 12 | Endepaket |
| PC | Pid_Ack_Byte | 6 | |

Hinweis: Die Pid_Ack_Byte bestätigen immer, dass zuletzt empfange Paket. Sollte einmal ein Fehler auftreten, werden sie durch ein Pid_Nak_Byte Paket ersetzt.

# 3. Benutzerhandbuch

## *3.1 Ablaufbedingungen*

Für den Programmablauf des GPS Daten Loggers werden folgende Hard- und Softwarekomponenten vorausgesetzt:

| Hardware |
|---|
| • Die aufgebaute Schaltung<br>• 5.0V Stromversorgung durch Netzteil oder Batterie<br>• Garmin GPS Handgerät<br>• Serielles Verbindungs-Kabel<br>• PC für Datensicherung und grafische Ansicht |

| Software |
|---|
| • Betriebssystem:<br>- Windows XP<br>• OZI-Explorer für die Übertragung der Way- und Trackpoints auf den PC und zur grafischen Darstellung der Punkte auf einer Landkarte |

## 3.2 Programmstart und Bedienungsanleitung

**Programmstart:**
Nach Aufbau der Schaltung und Anlegen der Betriebsspannung von 5.0V ist die Schaltung betriebsbereit. Die Statusanzeige signalisiert diesen Zustand durch „00“. Das Programm startet im GPS-Modus und ist bereit Daten an den PC zu übertragen (allerdings sind zu Beginn natürlich keine Way- oder Trackpoints gespeichert).

**PC-Betrieb:**
Durch Betätigen des Modus-Tasters kann die Schaltung in den PC-Betrieb wechseln. Es wird sofort eine Anfrage nach Way- dann nach Track-Points über die serielle Schnittstelle zum GPS-Handgerät gesendet. Dabei wird als erstes ein Handshake mit Protokollprüfung durchgeführt, was auf der Statusanzeige als „11“ zu erkennen ist. Anschließend werden zunächst alle im Handgerät vorhandenen Way-Points in das SRAM der Schaltung übertragen (Statusanzeige: „12“). Danach folgen die Track-Points (Statusanzeige: „13“). Ein erfolgreicher PC-Kommunikationsablauf schaltet die Statusanzeige auf „14“. Die Schaltung befindet sich wieder im Ausgangszustand (GPS-Modus).

**GPS-Betrieb:**
Ist die Schaltung z.B. nach dem Einschalten im GPS-Modus (Statusanzeige: „00“), kann ein angeschlossener PC die in der Schaltung gespeicherten Daten auslesen. Dazu muss vom PC aus eine Anfrage zur Schaltung losgeschickt werden, woraufhin ein Handshake mit Protokollprüfung durchgeführt wird (Statusanzeige: „01“). Ist der Verbindungsaufbau erfolgreich kann der PC entweder eine Anfrage nach Way- oder nach Track-Points stellen. Die Statusanzeige signalisiert die aktuelle Übertragung von Way-Points („02“) oder Track-Points („03“). Nach erfolgreicher Übermittlung der Daten befindet sich die Schaltung wieder im Ausgangszustand (GPS-Modus) und zeigt über die Statusanzeige den Abschluss an (Statusanzeige: „04“).

**Fehlerfälle (Timeout, Speicher voll):**
Tritt während einer Übertragung (egal ob im GPS- oder PC-Modus) ein Fehler auf (z.B. durch Kabelfehler), der nicht automatisch korrigiert werden kann, wird im GPS Daten Logger ein Timeout ausgelöst (Statusanzeige: „AA“). Die Schaltung wird dabei automatisch wieder in den Ausgangszustand zurückgesetzt (GPS-Modus). Es kann eine erneute Points-Anfrage vom PC erfolgen oder mittels Tastendruck ein angeschlossenes GPS-Handgerät ausgelesen werden.
Wird während einer Daten-Übertragung zum GPS Daten Logger festgestellt, dass der vorhandene Speicherplatz im SRAM nicht ausreicht, wird ein Fehler ausgelöst und der Status „88“ angezeigt. Die Übertragung wird abgebrochen und die gesamte Session wird nicht gesichert! Die Schaltung befindet sich anschließend wieder im GPS-Modus.

**Standby-Modus:**
Nach einem abgeschlossenen Übertragungsvorgang (im Ausgangszustand, GPS-Mode) empfiehlt es sich, die Schaltung in den Standby-Modus zu versetzen, indem der Standby-Schalter betätigt wird. In diesem Modus werden alle nicht benötigten Bausteine von der Stromversorgung getrennt. Lediglich der Speicherbaustein bleibt angeschlossen, damit die gespeicherten Daten nicht verloren gehen. Der Stromverbrauch sinkt jedoch auf ein Minimum ab und die Batterie wird nur minimal belastet. Soll eine erneute Übertragung oder ein Auslesevorgang gestartet werden, muss die Schaltung durch zurückstellen des Standby-Schalters wieder reaktiviert werden. Die Statusanzeige leuchtete wieder auf und bestätigt mit „23“ das „Aufwachen“ des GPS Daten Loggers, der sich wieder im Ausgangszustand (GPS-Modus) befindet.

### *3.3 Statusmeldungen*

Der GPS Daten Logger gibt (wie im vorigen Kapitel beschrieben) aktuelle Statusinformationen an den Benutzer aus. Folgende Tabelle zeigt eine Übersicht:

| **Beschreibung:** | **Modus (GPS-Gerät / PC):** | **Anzeige:** |
|---|---|---|
| Eingeschaltet | GPS | 00 |
| GPS Handshake | GPS | 01 |
| GPS Waypoints | GPS | 02 |
| GPS Trackpoints | GPS | 03 |
| GPS Übertragung fertig | GPS | 04 |
| PC Handshake | PC | 11 |
| PC Waypoints | PC | 12 |
| PC Trackpoints | PC | 13 |
| PC Übertragung fertig | PC → GPS | 14 |
| Timeout | GPS | AA |
| Speicher voll | GPS | 88 |
| Eingeschaltet nach Standby | GPS | 23 |

# 4. Hardwarehandbuch

## *4.1 Schaltplan*

### 4.1.1 Schaltungsplan

### 4.1.2 Stückliste

Die untenstehende Tabelle zeigt alle in der Schaltung verwendeten Elemente:

**Halbleiter:**
1 x Atmel 89C4051
1 x MAX 232N
1 x HY628400A SRAM
4 x 74LS273
1 x 74HC04
1 x 74LS08
2 x 74LS47
2 x 7-Segment (gem. Anode)

**Widerstände:**
2 x 100 kΩ
14 x 330 Ω

**Kondensatoren:**
2 x 33pF
5 x 2,2µF
6 x 100nF

**Sonstiges:**
1 x Quarz, 22,1184 MHz
1 x Drucktaster
1 x Schalter
1 x Duo-Schalter
1 x RS-232-Verbindungsstecker

## *4.2 Ansteuerung Atmel*

### 4.2.1 Interrupts

Der Atmel-Mikroprozessor unterstützt sowohl timergesteuerte Interrupts, die z.B. für die Nutzung der seriellen Schnittstelle genutzt werden können, als auch externe Interrupts, die durch Flanken oder Pegelwechsel an den entsprechenden Pins ausgelöst werden.

#### 4.2.1.1 Serielle Schnittstelle

Die im Atmel-Prozessor integrierte serielle Schnittstelle kann in vier verschiedenen Betriebsarten arbeiten, die über die Flags SM0 und SM1 bestimmt werden können. Unsere Schaltung benutzt Betriebsart 1 (SM0=false, SM1=true), bei der die serielle Schnittstelle im Asynchron-Betrieb mit 1 Startbit, 8 Datenbits und 1 Stoppbit arbeitet. Bei dieser asynchronen Datenübertragung wird keine eigene Taktleitung benötigt, Sender und Empfänger arbeiten weitgehend unabhängig voneinander. Um im Betriebsmodus 1 eine exakte Übertragungsrate von 9600 Baud zu erreichen, verwendet die Schaltung den internen 8-Bit Timer (TH0/TH1) des Atmel-Prozessors (interner Takt) mit Auto-Reload (M1=true, M0=false).

Die Baudrate lässt sich nach folgender Formel berechnen:

$$Baudrate = \frac{1}{16} \times \frac{Oszillatorfrequenz}{12 * (256 - (TH1))}$$ (SMOD=1, Zählerüberlauf geteilt durch 16)

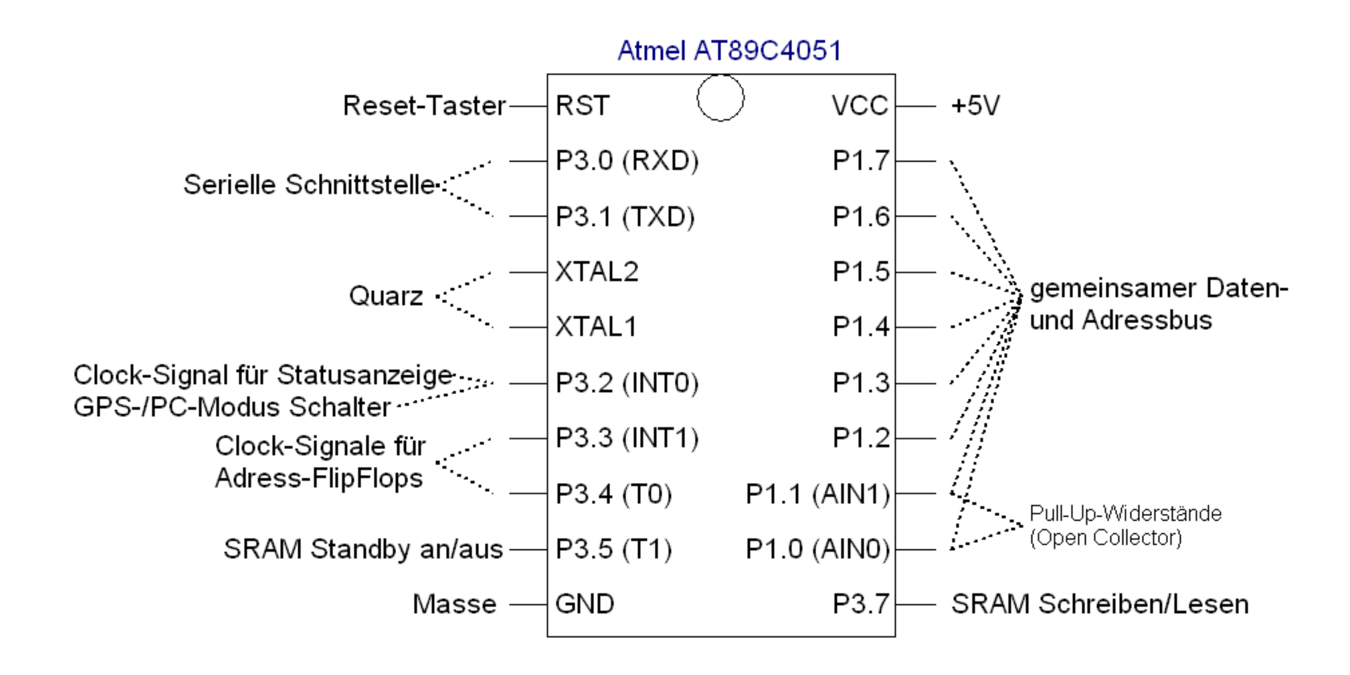
Durch den eingesetzten Quarz mit f=22,1184 MHz ergeben sich daraus folgende Werte:
TH0 =244, TH1 =100.

Für die Hardwareanschlüsse der seriellen Schnittstelle stellt der Atmel die beiden Pins P3.0 (RXD) für das Empfangen und P3.1 (TXD) für das Senden zur Verfügung.
Die Wandlung von den TTL-kompatiblen Pegeln des Atmels auf Pegel der seriellen Datenübertragung werden extern vom Treiberbaustein (MAX232N) vorgenommen (siehe unteren Abschnitt).
Dieser externe Baustein ist mit den RXD und TXD Pins des Atmels verbunden.

### 4.2.1.2 Externer Interrupt (Taster)

Der Atmel bietet über die Pins P3.2 (INT0) und P3.3 (INT1) zwei nach außen geführte Eingänge für externe Interrupts. Diese Eingänge sind Low-aktiv und können sowohl pegel- als auch flankengesteuert als Interrupt-Quellen arbeiten. Der Pegel wird dabei nach Auslösen in ein internes Latch(IEx) übernommen.
Unsere Schaltung verwendet einen Taster zum Wechsel vom „GPS-Modus“ in den „PC-Modus“ (um Daten zu empfangen). Dieser Taster ist mit dem INT0-Eingangspin des Atmels und der Masseleitung (logisch: Low) verbunden. Bei Tastendruck wird der Eingang auf Low „gezogen“ und ein Interrupt durch die Flanke (IT0=true: Interrupt arbeitet flankengesteuert) des Pegelwechsels ausgelöst.

## 4.2.2 Pinbelegung



## *4.3 Adressierung*

### 4.3.1 Beschreibung der Funktionalität des SRAMs

Für das Speichern der Way- und Trackpoints wird der HY628400A CMOS SRAM Baustein eingesetzt, der 512 Kbyte Speicherplatz zur Verfügung stellt. Der Chip verfügt über 19 Adresseingänge, an denen die gewünschte Speicherplatz-Adresse angelegt werden muss ($2^{19}$ entspricht 512K). Außerdem besitzt er über 8 Pins für ein Datenbyte, das entweder aus dem Speicher gelesen oder hinein geschrieben wird. Daneben gibt es noch drei Steuerbits, die das Verhalten des SRAM-Bausteins kontrollieren (Standby, Aktiv, Lesen, Schreiben). Die wichtigste Funktion für das Ein- und Ausschalten (Standby) des Speichers wird über den Pin /CS gesteuert, der in unserer Schaltung direkt mit dem Atmel-Pin P3.5 verbunden ist, so dass der Speicher über einfaches Setzen eines Pegels an diesem Ausgang sehr schnell aktiviert und deaktiviert werden kann. Darüber hinaus bestimmt ein weiterer Pin am Atmel (P3.7), ob vom Speicher gelesen (P3.7=true) oder in den Speicher geschrieben (P3.7=false) werden soll. Zu diesem Zweck ist der Pin direkt mit dem /WE (schreiben) und invertiert mit /OE (lesen) verschaltet. Ein Schreibevorgang in das SRAM besteht also immer aus folgenden Schritten:

1. Adresse an die Adressleitungen anlegen
2. Datenwort an das Datenbyte anlegen
3. Die Steuerbits zum Schreiben setzen (/WE = L)
4. Den SRAM aus dem Standby-Modus „wecken“ und aktiv schalten (/CS=L)
5. Durch das Aktivschalten des Chips wurde das anliegende Byte in das SRAM übertragen. Anschließend kann der SRAM wieder in den Standby-Modus versetzt werden (/CS=H)

### 4.3.2 Registerbausteine zur Adressierung

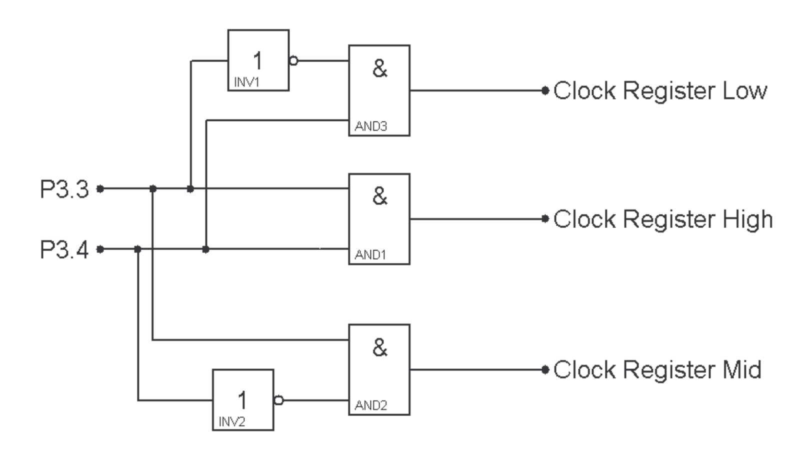
Das SRAM kann insgesamt 2 hoch 19 Bits (=512K) adressieren und abspeichern. Um diesen gesamten Bereich mit dem 8 Bit Ausgang des Atmel-Prozessors abzudecken sind dazu nacheinander drei Takte notwendig, die zunächst jeweils einen Teil der gesamten Adresse (Low, Middle, High) in je ein Register übertragen (Takt 1/2 übertragen je 8 Bit, Takt 3 überträgt 3 Bit). Dazu wird das entsprechende Adressbyte am Port1 des Atmels ausgegeben und die Taktleitung des Zielregisters eingeschaltet, so dass das Byte im Register übernommen und gespeichert wird. Alle drei Registerbausteine sind direkt mit dem Datenausgang des Atmels verbunden und können so das Byte empfangen. Es gibt einen gemeinsamen Adress- und Datenbus. Ist die Adresse vollständig in den drei FlipFlops abgelegt, kann das SRAM die komplette Adresse aus den FlipFlops der Registerbausteine beziehen, da diese auch direkt mit dem SRAM-Chip verbunden sind. Dazu wird der Chip-Select Eingang des SRAM-Bausteins auf Low gesetzt, was zur Folge hat, dass dieser den Standby-Modus verlässt und über die Bytes der drei Register die Adresse bestimmt und den Datenzugriff ermöglicht.

### 4.3.3 Logikschaltung zur „Taktung“ der FFs

Die einzelnen Registerbausteine mit den FlipFlops werden jeweils einzeln über eine Taktleitung aktiviert bzw. deaktiviert. Da jeweils nur ein Register zur Zeit zum Lesen aktiviert werden muss, gibt es vier Zustände für das Setzen der drei FlipFlops. Diese vier Zustände können mit einer simplen Logikschaltung abgebildet werden:

| **Zustand:** | **Binäre Codierung: P3.3 und P3.4 am Atmel** | **Logik für die Taktleitungen der Registerbausteine:** |
|---|---|---|
| Zustand 1: | 0 0 | Alle Register aus (aktuelle Adresse in den drei Registern bleibt unverändert): $R_{aus} = \overline{A} \wedge \overline{B}$ |
| Zustand 2: | 0 1 | Nur das Register für Low-Anteil an: $R_1 = \overline{A} \wedge B$ |
| Zustand 3: | 1 0 | Nur das Register für Mid-Anteil an: $R_2 = A \wedge \overline{B}$ |
| Zustand 4: | 1 1 | Nur das Register für High-Anteil an: $R_3 = A \wedge B$ |

Die benötigten Zustände können mit Hilfe von zwei Invertern und drei Und-Verknüpfungen gebildet werden:




### 4.3.4 Beispiel für einen Adressierungsvorgang

Im folgenden Beispiel soll ein Byte (8 Bit) an die Speicherzelle mit der Adresse 2342 geschrieben werden. Die Adresse 2342 (dezimal) wird intern wie unten beschrieben binär abgebildet:

```
High       Mid        Low
00000000 | 00001001 | 00100110 (binary)

Die einzelnen Anteile sind:
       0 |        9 |       38 (dezimal)
```

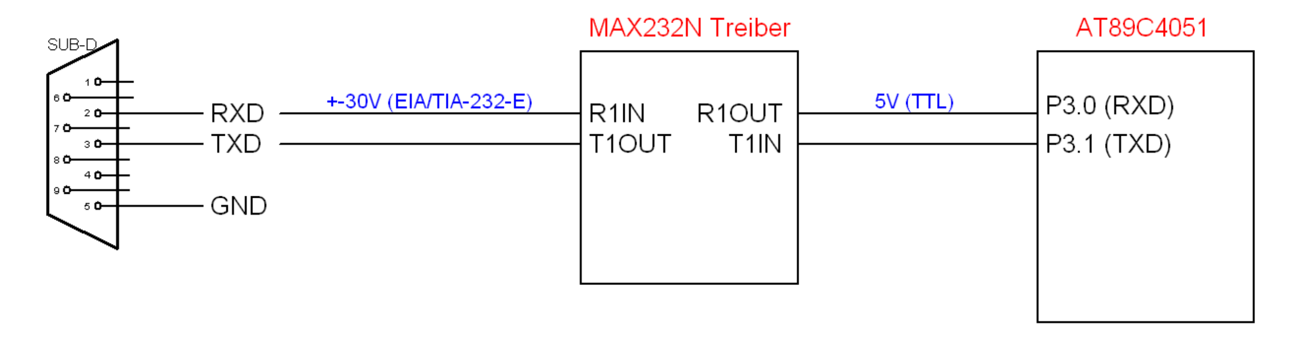
Da die Schaltung einen gemeinsamen Adress- und Datenbus von 8-Bit Breite verwendet, sind dazu vier Schritte notwendig:

1. Zunächst wird am Datenausgang Port1 des Atmels der High-Anteil der Adresse (hier dezimal: 0) angelegt (da wir nur $2^{19}$ Adressen verwalten können sind hier nur die letzten 3 Bits entscheidend). Anschließend wird mit Hilfe zwei weiterer Atmel-Pins (P3.3 und P3.4) und der Logikschaltung das Register für den High-Anteil aktiviert, so dass die Register-FlipFlops das am Port1 angelegte Byte speichern und das erste Adressbyte gesetzt ist. Anschießend wird der Clock-Eingang des Register durch die beiden Pins des Atmels (P3.3 und P3.4) wieder zurückgezogen, so dass Veränderungen an Port1 für das Register nicht mehr relevant sind.

2. Als nächstes wird das zweite Byte der Adresse (hier dezimal: 9) an Port1 des Atmels gesetzt. Dann wird mit Hilfe der beiden Steuer-Pins (P3.3 und P3.4) der Clock-Eingang des zweiten Registers aktiviert, woraufhin das Byte in den Register-FlipFlops übernommen wird. Ist der Wert gespeichert, muss das Register durch Wiederentfernen des Clock-Signals (über P3.3 und P3.4) wieder vom Port1 entkoppelt werden.
3. Im dritten Schritt wird schließlich das letzte Adressbyte (hier dezimal: 38) am Port1 des Atmels ausgegeben und über die Steuerpins wie bereits zuvor in das dritte Register gespeichert. Auch hier wird das Clock-Signal des Register danach zurückgezogen, so dass nun alle Registerbausteine zusammengenommen die Adresse 2342 enthalten und vom Port1 des Atmels entkoppelt sind.
4. Im letzten Schritt kann nun das eigentliche Datenbyte am Port1 des Atmels angelegt werden. Das SRAM ist über zwei Pins (P3.5 und P3.7) direkt mit dem Atmel verbunden. Dadurch kann jetzt über entsprechende Pegel der Schreibmodus des SRAM-Bausteins gesetzt und gleich darauf der Standby-Modus des Chips verlassen werden. Dies bewirkt, dass der aktuelle Wert von Port1 des Atmels an der Adresse im SRAM abgespeichert wird, die durch die drei Register anliegt. Zum Schluss wird der SRAM-Baustein wieder in den Standby-Modus geschaltet, so dass Veränderungen am Datenport des Atmels keine Auswirkungen mehr auf den Speicherchip haben.

## 4.4 Datenkommunikation (RS-232, MAX232N Baustein)

Die Kommunikation mit der seriellen Schnittstelle wird direkt durch den Atmel-Prozessor unterstützt und gesteuert. Da auf physikalischer Ebene jedoch mit unterschiedlichen Spannungspegeln auf TTL- (Atmel/Schaltung) und RS-232-Seite gearbeitet wird, ist eine Umsetzung durch einen Treiberbaustein notwendig. Diese Aufgabe übernimmt der MAX232N-Baustein. Er ist dabei zwischen den RXD/TXD Ports des Atmels und dem Anschluss der seriellen Schnittstelle geschaltet, wie folgende Grafik veranschaulicht:




## 4.5 Statusanzeige über 7-Segment-Display

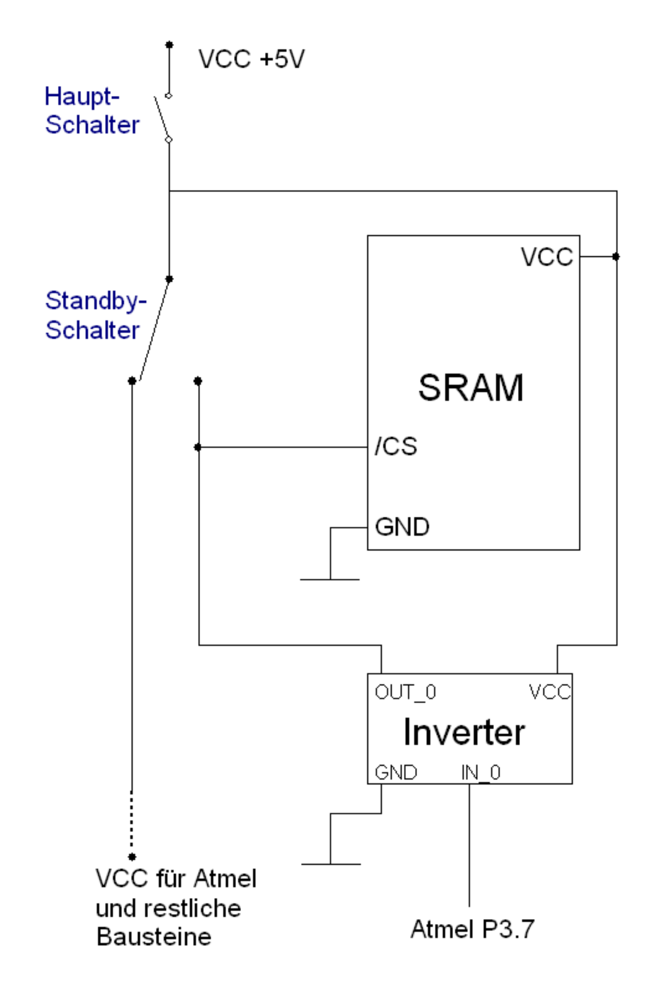
Damit die Schaltung Informationen an den Benutzer ausgeben kann, ist eine Statusanzeige implementiert. Diese besteht aus zwei 7-Segmentbausteinen die jeweils an einem BCD-7seg Konverter (7447) angeschlossen sind. So kann jede der beiden Anzeigen eine Hexadezimalzahl (0-F) ausgeben. Die Konverter-Bausteine haben jeweils vier Eingabe-Pins, die indirekt aus dem Datenbus des Atmel-Prozessors gespeist werden. Der Datenbus ist jedoch nicht direkt angeschlossen, sondern über ein Register-Baustein mit 8 FlipFlops verschaltet. So kann der Datenbus des Atmels von der Anzeige entkoppelt werden und muss nur zum Aktualisieren der Anzeige die Verbindung herstellen, indem das Register über die Taktleitung aktiviert wird.

Die Konverter codieren jeweils 4 Bits aus dem Register für eine 7-Segment-Anzeige um und geben den neuen Zustand an den Ausgängen aus, so dass der gewünschte Hex-Wert auf dem Display erscheint.

## *4.6 SRAM-Standbymodus*

Ein wichtiger Gesichtspunkt beim Entwurf der Schaltung ist die Stromaufnahme. Da die Schaltung nicht über das Netz sondern durch Batterien gespeist werden soll, ist ein geringer Stromverbrauch eine grundlegende Bedingung, um längere „Sessions“ des Anwenders ohne Datenverlust zu ermöglichen. Ein möglicher Ansatz wäre, nach erfolgreicher Datenübertragung den Atmel- und SRAM-Baustein in den unterstützen Standby-Modus zu setzen und die 7-Segment-Anzeige auszuschalten, was den Stromverbrauch bereits deutlich verringern würde. Jeder TTL-Baustein der Logik würde jedoch noch ca. 10 mA Strom aufnehmen und auch der Atmel-Prozessor fordert seinen Anteil.
Unser Entwurf sieht deshalb vor, dass der Benutzer nach erfolgreicher Übertragung über einen zusätzlichen Schalter die Möglichkeit bekommt, nahezu die gesamte Hardware vom Stromkreis zu trennen. Lediglich der wichtige, datenerhaltende SRAM-Speicher und ein zugehöriger Inverter-Baustein werden am Leben erhalten und nicht von der Batterie entkoppelt. Dadurch sinkt die Stromaufnahme der Schaltung auf ein Minimum. Die folgende Tabelle verdeutlicht den Gewinn:

| **Modus:** | **Stromaufnahme:** |
|---|---|
| Gesamte Schaltung eingeschaltet (inkl. 7-Seg-Anzeige) | 210 mA |
| Gesamte Schaltung eingeschaltet (ohne 7-Seg-Anzeige) | 110 mA |
| Standby (nur SRAM und Inverter aktiv) | 4,5 µA !!! |




Die linke Abbildung stellt die Realisierung des Standby-Modus mit Hilfe eines zusätzlichen Duo-Schalters dar.
Durch Umlegen des Standby-Schalters wird der Chip-Select-Eingang des SRAM-Bausteins auf High-Pegel gezogen und der SRAM somit in Standby versetzt, wo er auf keine Daten-Signale mehr reagiert. Gleichzeitig ist die Vcc-Versorgung der restlichen Bausteine getrennt (bis auf den Inverter). Wird der Schalter anschließend zurück in die gezeigte Ausgangslage gestellt, sind alle Elemente wieder mit Spannung versorgt und der Atmel-Prozessor kann erneut „hochfahren“. Durch einen Checksummen-Test der ersten Bytes im SRAM erkennt der Chip das Erwachen aus dem Standby-Modus, woraufhin er alle internen Variablen aus dem SRAM bezieht und keine Neuinitialisierung durchführt (siehe Kap. 5.2.7).

# 5. Programmierhandbuch

## 5.1 Entwicklungskonfiguration

Die Entwicklung des GPS Loggers wurde mit Hilfe folgender Konfiguration durchgeführt:

**Hardware**
- Standard-PC (WinXP kompatibel)
- Serielle Schnittstelle (RS-232)
- Programmer für Atmel AT89C4051
- 5.0V Netzteil, Steckplatine, Bausteine, Kabel etc.
- Garmin GPS Handgerät

**Software**
- Betriebssystem: Windows XP
- Programmierumgebung: System51-Compiler, SciTE Editor
- Windows Hyper Terminal
- Docklight (zum Testen der seriellen Kommunikation Testen)
- OZI Explorer (zum Auslesen der Way- und Trackpoints)

## 5.2 Problemanalyse und Realisation

### 5.2.1 Grundsätzlicher Programmaufbau und Programmgliederung

Der Softwareteil des Projektes „GPS-Datenlogger“ kann nach folgenden Gesichtpunkten gegliedert werden:
- Aufbau der Mainloop und Interruptverarbeitung
- Datenkommunikation über die serielle Schnittstelle
- Statusanzeige auf der 7-Segment Anzeige
- Speicherverwaltung und Datenkompression
- Datenverarbeitung auf Byte-Ebene
- Datenverarbeitung auf Paketebene
- Realisierung einer Standbyschaltung
- Timeout Überprüfung

Diese Punkte werden in den folgenden Abschnitten ausführlich erläutert.

### 5.2.2 „Mainloop“ und Interruptverarbeitung

Wie in der Mikrocontrollerprogrammierung meist üblich, wird auch hier, nach der Initialisierungsphase, der Prozessor in eine Endlosschleife, die Mainloop, versetzt. Somit ist grundsätzlich ein dauerhafter Programmablauf gewährleistet.
Wird ein Interrupt ausgelöst, wird die jeweilige Interrupt-Service-Routine *(ISR)* abgearbeitet. Die jeweiligen ISR’s selber sind vom Prinzip her ziemlich einfach aufgebaut. In diesen wird quasi nur ein Flag gesetzt, dass der jeweilige Interrupt aufgetreten ist. Diese Flags werden bei jedem Durchlauf der Mainloop abgefragt. Ist ein Flag gesetzt, werden entsprechende Routinen aufgerufen, die dann für die eigentliche Funktionalität des Programms enthalten.
In diesem Programm sind folgende zwei Interrupts freigegeben: Der Interrupt für die serielle Schnittstelle und der externe Interrupt 0. Der externe Interrupt 0 wird nach einem Tastendruck ausgelöst und schaltet den PC-Modus des Gerätes ein. Der Interrupt der seriellen Schnittstelle wird dann ausgelöst, nachdem ein Byte über die diese empfangen oder gesendet worden ist.

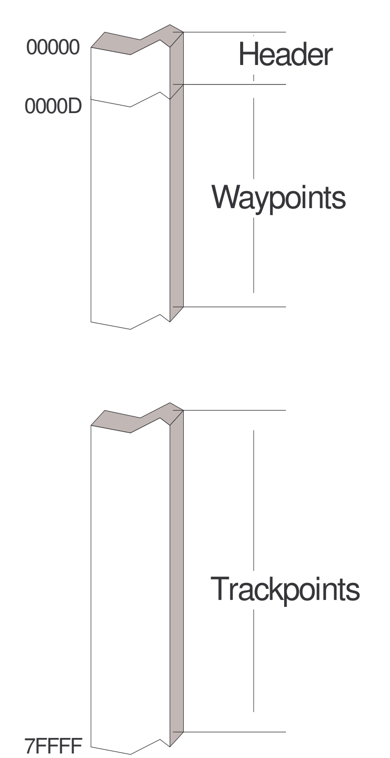
## 5.2.3 Datenübertragung über die serielle Schnittstelle

Wie oben schon erwähnt, wird *nach* dem Senden oder Empfangen eines Bytes ein Interrupt ausgelöst. Um zu unterscheiden, ob ein Byte empfangen oder versendet worden ist, werden die Bits „RI“ (Recieve-Interrupt) und „TI“ (Transmit-Interrupt) verwendet: Wurde ein Byte empfangen, wird RI gesetzt, Wurde ein Byte versendet, wird TI gesetzt. Die jeweiligen Bits müssen jedoch manuell wieder zurückgesetzt werden, was am Ende der ISR geschieht.
Soll ein Byte versendet werden, so muss der zu versendende Bytewert der Variablen „sbuf“ zugewiesen werden. Nach der Zuweisung wird automatisch mit der Versendung begonnen. Sollen mehrere Bytes hintereinander gesendet werden, muss gewährleistet sein, dass das Byte komplett versendet worden ist, bevor „sbuf“ ein neuer Wert zugewiesen wird. Dafür wird direkt nach der Zuweisung auf „sbuf“ ein Sendeflag gesetzt. Vor der Zuweisung des nächsten Bytes wird so lange gewartet, bis das Flag wieder zurückgesetzt worden ist, was mit Beendigung der ISR geschieht.
Wurde ein Byte empfangen, steht nach dem Interrupt der empfangene Byte-Wert in der Variablen „sbuf“. Dieser Wert wird anschließend im Programm weiterverwendet. Nach dem Empfang eines Bytes wird immer das Flag gesetzt, das in der Mainloop abgefragt wird, um es im Programm weiterzuverarbeiten.

## 5.2.3 Statusanzeige auf der 7-Segment Anzeige

Um ein Statusbyte auf der 7-Segment Anzeige auszugeben, wird dieses Byte an Port 1 des Prozessors angelegt. Anschließend wird das Statusclock-Bit gesetzt, was bewirkt, dass das Byte in das Register des Statusbytes übernommen wird. Anschließend wird das Bit wieder zurückgesetzt.

## 5.2.4 Speicherverwaltung und Kompression



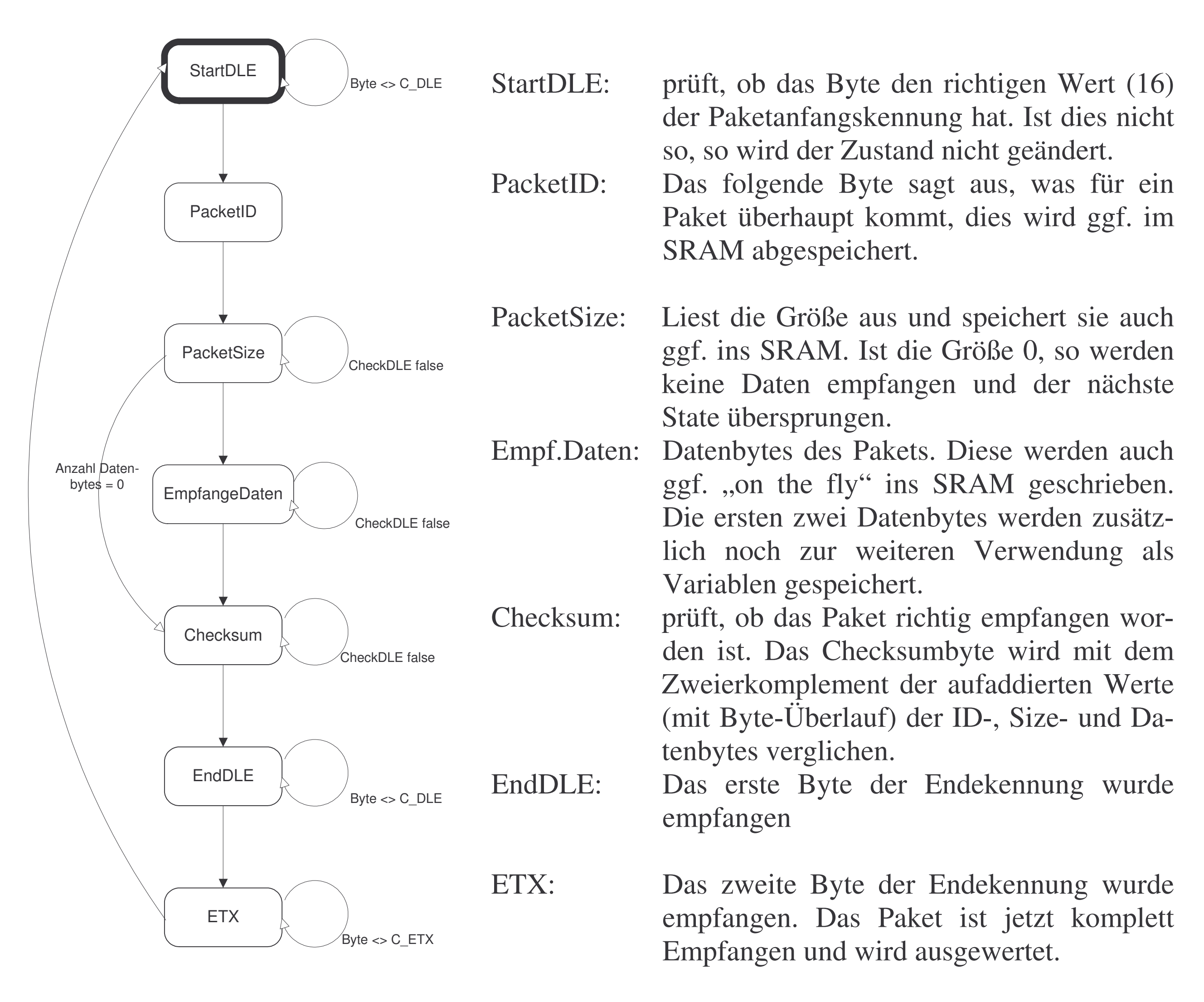

Die Daten, die abgespeichert werden müssen, bestehen aus den Waypoints und Trackpoints der GPS-Informationen. Die Speicherplatz des SRAMS wird so vergeben, dass am Anfang des Speichers die Informationen für die Wakeup-Funktion abgelegt werden (siehe dazu auch 5.2.7). Dahinter folgen dann die Bytes der Waypoints. Die Trackpoints werden ab dem Ende aufsteigend gespeichert. Das Nebenstehende Bild veranschaulicht die Speicherbelegung weitestgehend.
Für jedes einzelne Paket werden nur die PacketID, die Anzahl der Datenbytes und die Datenbytes selber abgespeichert. Daher, dass die Start- und Endebytes eines Paketes nicht mit abgespeichert werden, ist quasi eine kleine Datenkompression zustandegekommen. Unten sei nocheinmal der Aufbau eines Paketes im SRAM-Speicher bildlich dargestellt.

| 1 Byte ID | 1 Byte Size | n Bytes Daten |
| --- | --- | --- |
| | n | |

## 5.2.5 Datenverarbeitung auf Byte-Ebene

### 5.2.5.1 Empfang eines Paketes

Beim Datenempfang wird, wie oben erwähnt, nach jedem Byte ein Interrupt ausgelöst, was zur Folge hat, dass die Routinen aufgerufen werden, die das Byte verarbeiten können. Beim nächsten Byte werden die Routinen jedoch wieder von neuem aufgerufen. Eine kontinuierliche Datenverarbeitung beim Empfang über mehrere Bytes ist somit nicht möglich. Daher bietet sich hier der Einsatz von Statemachines an. Nach jedem Empfang eines Bytes wird also eine Rountine aufgerufen, die eine Statemachine repräsentiert. Das Zustandsdiagramm ist im folgenden Bild zu sehen. Rechts die Erläuterung der jeweiligen States für den Empfang eines Paketes.




| | |
|---|---|
| StartDLE: | prüft, ob das Byte den richtigen Wert (16) der Paketanfangskennung hat. Ist dies nicht so, so wird der Zustand nicht geändert. |
| PacketID: | Das folgende Byte sagt aus, was für ein Paket überhaupt kommt, dies wird ggf. im SRAM abgespeichert. |
| PacketSize: | Liest die Größe aus und speichert sie auch ggf. ins SRAM. Ist die Größe 0, so werden keine Daten empfangen und der nächste State übersprungen. |
| Empf.Daten: | Datenbytes des Pakets. Diese werden auch ggf. „on the fly“ ins SRAM geschrieben. Die ersten zwei Datenbytes werden zusätzlich noch zur weiteren Verwendung als Variablen gespeichert. |
| Checksum: | prüft, ob das Paket richtig empfangen worden ist. Das Checksumbyte wird mit dem Zweierkomplement der aufaddierten Werte (mit Byte-Überlauf) der ID-, Size- und Datenbytes verglichen. |
| EndDLE: | Das erste Byte der Endekennung wurde empfangen |
| ETX: | Das zweite Byte der Endekennung wurde empfangen. Das Paket ist jetzt komplett Empfangen und wird ausgewertet. |

Wurde ein Track- oder Waypoint empfangen und die Checksumme war falsch, so wird der aktuelle Adresszeiger wieder auf das erste Byte nach dem letzten gespeicherten Pakets zurückgesetzt. Anschließend werden, je nach Modus die Routinen zur Verarbeitung und Auswertung der Daten auf Paketebene aufgerufen.

### 5.2.5.2 DLE-Wert Überprüfung

Das Protokoll sieht eine Besonderheit vor: Kommt in den ID-, Größen-, Daten- oder Checksumbytes ein Byte mit dem wert des DLE's (16) vor, so wird dieses verdoppelt. Dies muss natürlich in der Statemachine berücksichtigt werden. In jedem der relevanten States wird geprüft, ob ein solches Byte empfangen worden ist. Ist dies der Fall, so muss das folgende Byte ja ebenfalls diesen Wert haben. Da dies auch nicht für die Berechnung der Checksumme relevant ist, kann dieses quasi „übersprungen" werden.

### 5.2.5.3 Speicherüberlaufprüfung

Vor jedem Schreibzugriff auf das SRAM wird außerdem eine Speicherüberlaufsprüfung vorgenommen. Da ja die Waypoints ab dem Speicheranfang und die Trackpoints ab dem Speicherende abgelegt werden, kann der Adresszeiger für die Waypoints nicht größer sein als der für die Trackpoints. Also werden diese beiden Adressen vor dem Schreiben miteinander verglichen.
Ist der Speicher voll, so wird das aktuelle und alle bisherigen gespeicherten Pakete dieser Session verworfen.

### 5.2.5.2 Versenden eines Paketes

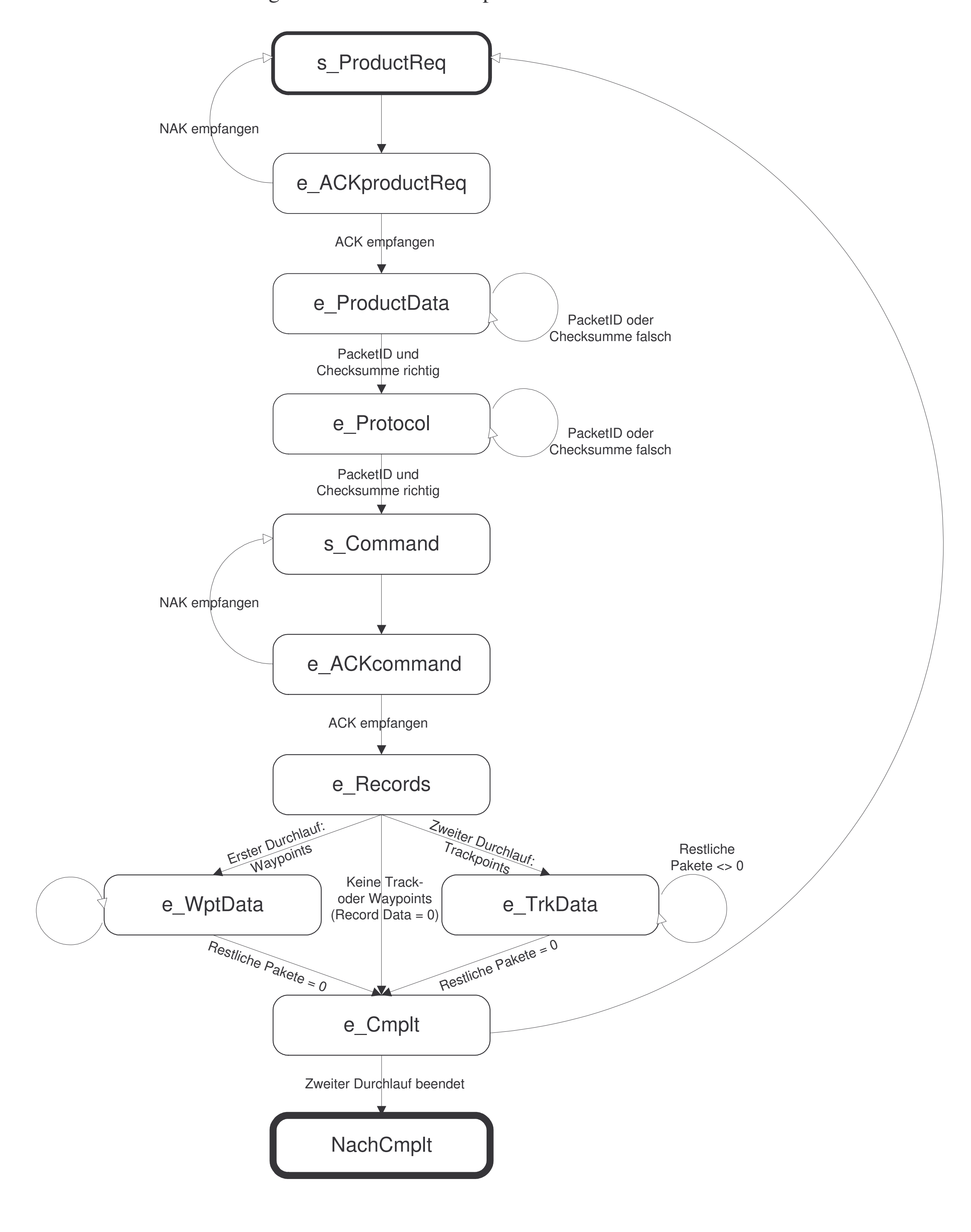
Das Versenden eines Pakets ist nicht so sehr komplex wie das Empfangen. Die Pakete, die für die Handshake-Zustände benötigt werden, sind als Konstanten im Programmcode abgelegt. Diese werden einfach byteweise mithilfe einer Schleife versendet.
Soll ein Track- oder Waypoint versendet werden, so müssen die Bytes erst aus dem SRAM ausgelesen werden. Hierbei ist noch zu beachten, dass vor dem Lesezugriff auf das SRAM alle Bits von Port 1 auf „1" gesetzt werden müssen. Ansonsten verhält sich der Vorgang ähnlich.

### 5.2.6 Datenverarbeitung auf Paketebene

Immer wenn ein Paket komplett empfangen worden ist, oder das Paketsende-Flag gesetzt ist, wird eine der beiden Hauptroutinen zur Paketverarbeitung aufgerufen. Auch hier ist das Konzept der Statemachine ideal. Falls im Folgezustand ein Paket versendet werden soll, wird das Paketsende-Flag gesetzt, welches in der Mainloop abgefragt wird. Dies ist nötig, da ja, im Gegensatz zum Paketempfang, kein Interrupt ausgelöst wird, mit dem die Datenverarbeitung fortgesetzt werden kann.
Wenn ein Paket fehlerhaft ist (falsche Checksumme oder falsche PackedID) wird dieses Paket ignoriert
Je nach aktuellem Betriebsmodus ist eine der beiden folgenden Statemachines aktiv.

### 5.2.6.1 PC-Modus

Für den PC-Modus ist folgende Statemachine implementiert:

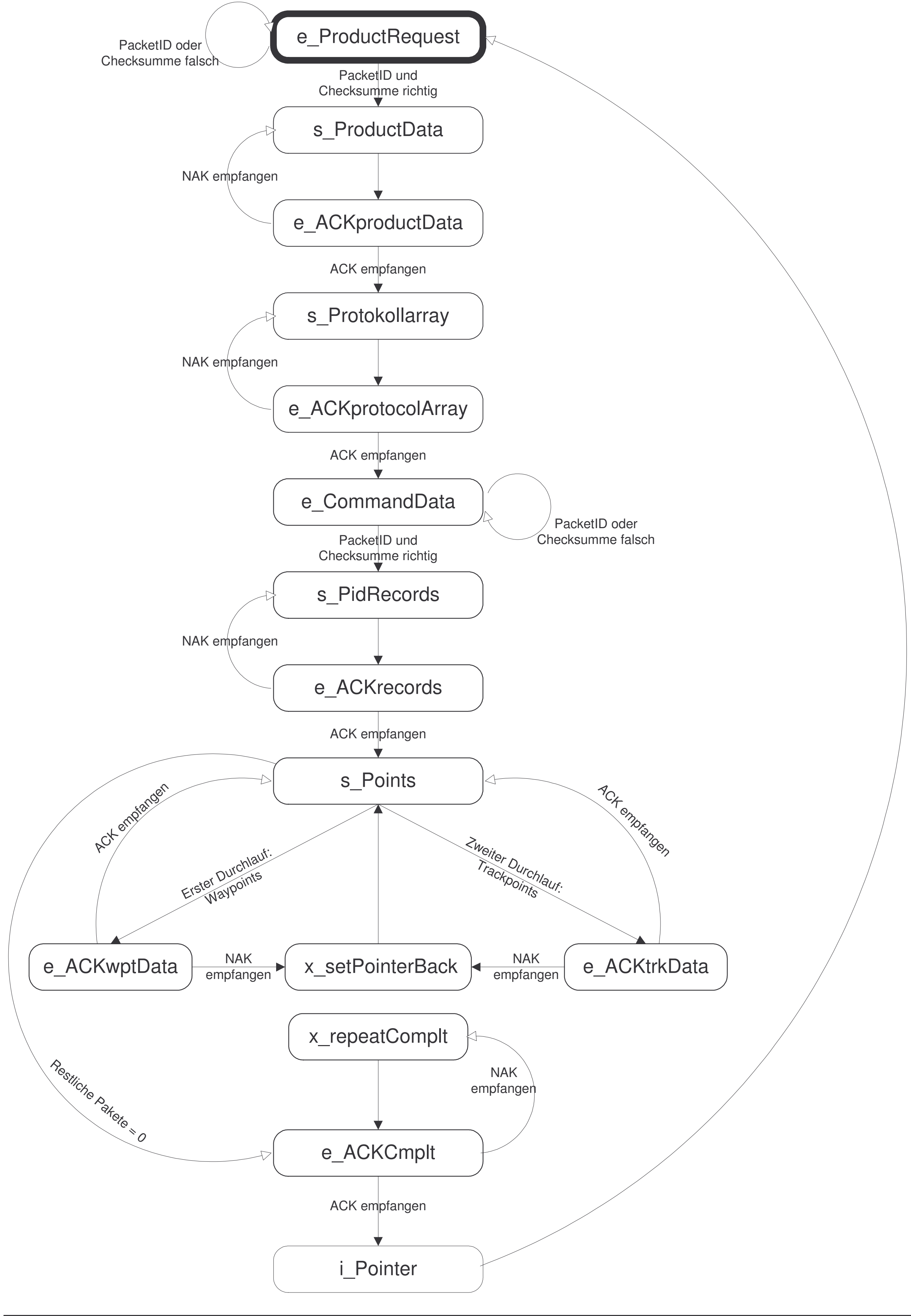


**Erläuterung der einzelnen States:**

| | |
|---|---|
| s_ProductReq | Dies ist der Startzustand der Statemachine. Es wird das Product-Request-Paket an das GPS-Gerät gesendet. Dies ist als Konstante im Programmcode enthalten. |
| e_ACKprodReq | Es wird eine Bestätigung für das Product-Request-Paket erwartet. Wird ein NAK empfangen, wird in den Vorigen Zustand zurückgesprungen, bei einem ACK wird normal fortgefahren. |
| e_ProductData | Das Paket mit den Produktdaten wurde empfangen. Hierbei wird nur noch die PacketID überprüft. Die Daten als solches haben für den Programmablauf keine Relevanz mehr. Bei gültigem Paket wird mit einem ACK bestätigt. Bei ungültigem Paket wird mit NAK bestätigt und der Zustand beibehalten. |
| e_Protocol | Das Paket mit den Protokolldaten wurde empfangen. Hierbei wird nur noch die PacketID überprüft. Die Daten als solches haben für den Programmablauf keine Relevanz mehr. Bei gültigem Paket wird mit einem ACK bestätigt. Bei ungültigem Paket wird mit NAK bestätigt und der Zustand beibehalten. |
| s_Command | Die Kommandodaten werden gesendet. Beim ersten Durchlauf wird der Request für die Waypoints im Datenanteil gesendet, beim zweiten Durchlauf der Request für die Trackpoints. |
| e_ACKcommand | Es wird eine Bestätigung für das Command-Data-Paket erwartet. Wird ein NAK empfangen, wird in den Vorigen Zustand zurückgesprungen, bei einem ACK wird normal fortgefahren. |
| e_Records | Das Paket mit der Anzahl der folgenden Way-, bzw. Trackpoints wurde empfangen. Da ja in der Byte-Ebene immer die ersten beiden Datenbytes global abgespeichert worden sind, ist jetzt in diesen Variablen die Anzahl der folgenden Way-/Trackpoints gespeichert. Somit kann jetzt der Zähler für die Way-/Trackpoint-Pakete initialisiert werden. Ist die Anzahl der Pakete 0, so wird gleich das X-fer-Complete Paket erwartet, es wird also in den e_Complt-Zustand gewechstelt. Ansonsten wird beim ersten Durchlauf in den State zum Empfangen der Waypoints gewechselt, beim zweiten in den State für die Trackpoints. |
| e_WptData | Die Waypoints werden empfangen. Ist die Anzahl der noch folgenden Pakete 0, so wird ein X-fer-Complete Paket erwartet, es wird also in den e_Complt-Zustand gewechstelt. Ansonsten wird in diesem State geblieben. Wenn das Paket gültig ist, wird der Zähler für die restlichen Pakete dekrementiert und mit einem ACK bestätigt. Ansonsten wird ein NAK versendet. |
| e_TrkData | Die Trackpoints werden empfangen. Dabei können Trackpoint-Header oder Trackpoint-Daten empfangen werden. Diese werden jedoch gleich behandelt und spielen im weiteren Verlauf keine Rolle. Ist die Anzahl der noch folgenden Pakete 0, so wird ein X-fer-Complete Paket erwartet, es wird also in den e_Complt-Zustand gewechstelt. Ansonsten wird in diesem State geblieben. Wenn das Paket gültig ist, wird der Zähler für die restlichen Pakete dekrementiert und mit einem ACK bestätigt. Ansonsten wird ein NAK versendet. |
| e_Complt | Das X-fer-Complete Paket wurde empfangen. Nach Beendigung des ersten Durchlaufs wird in den s_ProductReq-Zustand zurückgesprungen, um sofort anschließend den Transfer der Waypoints zu starten. Ist der zeite Durchlauf beendet, so ist der Datentransfer abgeschlossen und es wird in den NachCmplt-Status gewechselt |
| NachCmplt | Dieser Zustand repräsentiert einen Standbyzustand. Dieser ist dazu geeignet, um eventuellen „Datenmüll“ zu ignorieren, da aus diesem Zustand nicht herausgesprungen werden kann. Dies kann nur durch erneuten Tastendruck geschehen, da in der zugehörigen Bearbeitungsroutine die Statemachine neu initialisiert wird. |

### 5.2.6.2 GPS-Modus

Für den PC-Modus ist folgende Statemachine implementiert:



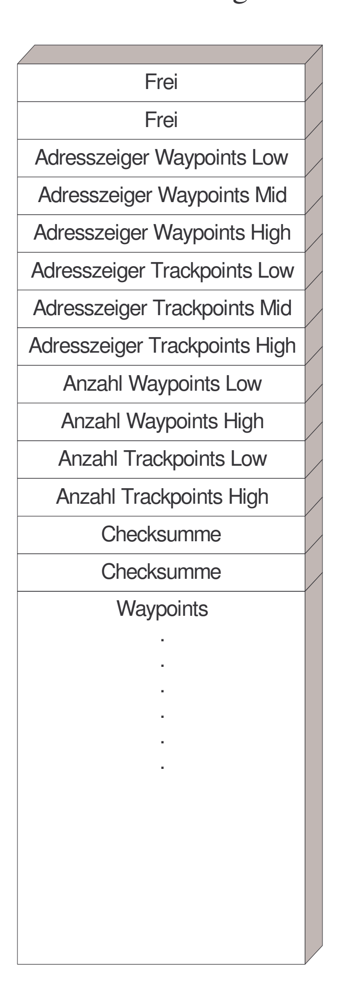
**Erläuterung der einzelnen States:**

| | |
|---|---|
| e_ProductRequest | Da sich das Gerät nach dem Einschalten sofort im GPS-Modus befindet, wird sofort, nachdem ein gültiger Productrequest empfangen worden ist, in den nächsten Zustand gewechselt. Die Gültigkeit wird neben der Checksumme auch mit der PacketID geprüft. |
| s_ProductData | Die Produktdaten befinden sich als Konstanden im Programmcode und werden an den PC versendet. |
| e_ACKproductData | Es wird eine Bestätigung für das Product-Data-Paket erwartet. Wird ein NAK empfangen, wird in den Vorigen Zustand zurückgesprungen, bei einem ACK wird normal fortgefahren. |
| s_Protokollarray | Die Daten des Protokollarrays befinden sich als Konstanden im Programmcode und werden an den PC versendet. |
| e_ACKprotokollArray | Es wird eine Bestätigung für das Protokoll-Array-Paket erwartet. Wird ein NAK empfangen, wird in den Vorigen Zustand zurückgesprungen, bei einem ACK wird normal fortgefahren. |
| e_CommandData | Das Kommandopaket wird empfangen. In den Datenbytes ist festgelegt, ob Waypoints oder Trackpoints gesendet werden sollen. Das Paket wird bei Gültigkeit mit ACK bestätigt, sonst mit NAK. |
| s_PidRecords | Im Programm sind Variablen enthalten, die die aktuelle Anzahl von Waypoints und Trackpoints im SRAM enthalten. Die Werte der relevanten Variablen (Way- oder Trackpoints) werden als Datenbytes im Little-Endian Format in diesem Paket versendet. |
| e_ACKrecords | Es wird eine Bestätigung für das Records-Paket erwartet. Wird ein NAK empfangen, wird in den Vorigen Zustand zurückgesprungen, bei einem ACK wird normal fortgefahren. |
| s_Points | Es wird das aktuelle Paket aus dem Speicher gelesen und versendet, der Adresszeiger wird dabei aktualisiert und der Zähler für die noch zu versendenden Pakete dekrementiert. |
| x_setPointerBack | Falls das zuletzt gesendete Paket nicht richtig übertragen worden ist, also mit einem NAK bestätigt worden ist, muss es noch einmal verschickt werden. Dafür muss der Adresszeiger auf das vorige Paket zurückgesetzt werden und der Zähler für die noch zu verschickenden Pakete um 1 inkrementiert werden, was in diesem Zustand geschieht. |
| e_ACKwptData | Es wird eine Bestätigung für ein Waypoint-Paket erwartet. Wird ein NAK empfangen, wird in den x_setPointerBack Zustand gesprungen, bei einem ACK wird normal fortgefahren. Sind alle Pakete versendet worden wird als nächstes ein X-fer-Complete-Paket erwartet, ansonsten wird wieder in den s_Points Zustand gewechselt |
| e_ACKtrkData | Es wird eine Bestätigung für ein Trackpoint-Paket erwartet. Wird ein NAK empfangen, wird in den x_setPointerBack Zustand gesprungen, bei einem ACK wird normal fortgefahren. Sind alle Pakete versendet worden wird als nächstes ein X-fer-Complete-Paket erwartet, ansonsten wird wieder in den s_Points Zustand gewechselt |
| x_repeatComplt | Das X-fer-Complete-Paket ist nicht richtig versendet worden, was zur Folge hat, dass dieses nocheinmal versendet werden muss. |
| e_ACKComplt | Es wird eine Bestätigung für das X-fer-Complete-Paket erwartet. Wird ein NAK empfangen, wird in den Zustand x_repeatComplt gesprungen, bei einem ACK ist der Transfer abgeschlossen und es wird in den i_pointer Zustand gewechselt |
| i_Pointer | Alle Zähler und Adresszeiger der Way- oder der Trackpoints werden neu initialisiert. Die soeben transferierten Daten werden also quasi „gelöscht“. |

### 5.2.7 Standbyrealisierung

Um den Standbybetrieb zu realisieren sind folgende Dinge notwendig:

- Abspeichern der Adresszeiger und die Anzahl der Track- und Waypoints im SRAM abspeichern.
- Das Erkennen beim Prozessorstart, ob aus dem Standbybetrieb gestartet wird oder ob ein Kaltstart erfolgt ist.




Die Adresszeiger auf das erste freie Byte nach den Waypoints (3 Bytes), der Adresszeiger auf das erste freie Byte nach den Trackpoints (3 Bytes), die Anzahl der Waypoints im Speicher (2 Bytes) und die Anzahl der Trackpoints im Speicher (2 Bytes) werden im reservierten Speicherbereich (Header) nach jedem Abgeschlossenen Datentransfer abgespeichert. Anschließend wird aus diesen 10 Bytes eine Checksumme errechnet, die zweimal hinter diesen 10 Bytes abgelegt wird. Der Header des SRAMS ist links dargestellt. Die Ckecksumme wird folgendermaßen berechnet: Alle Bytes werden Addiert (mit Byteüberlauf) und anschließend wird von dieser Summe das Zweierkomplement gebildet.

Bei jedem neuen Prozessorstart wird der Header ausgelesen und die Checksumme der 10 relevanten Datenbytes mit den beiden Checksummenbytes aus dem Speicher verglichen. Stimmen alle drei Bytes überein, werden die Adresszeiger und Paketzähler automatisch gesetzt.
Wenn der Speicher eingeschaltet wird, ist dessen Inhalt undefiniert. Dadurch, dass sie Checksumme zweimal abgespeichert wird ist die Wahrscheinlichkeit, dass trotz eines Kaltstarts alle drei zu vergleichenden Bytes übereinstimmen auf ein Minimum reduziert. Wenn nur ein Byte als Checksumme abgespeichert werden würde, läge die Wahrscheinlichkeit noch bei 1:255, was ein inakzeptabler Wert wäre. Bei zwei Bytes beträgt die Wahrscheinlichkeit jedoch schon bei 1:65535, was einen guten Wert darstellt.

### 5.2.8 Timeout Überprüfung

Kommt es während des Datentransfers zu einem Kommunikationsfehler (z.B. Kabel verliert Kontakt), würde das Gerät im aktuellen Zustand bleiben und müsste durch einen kompletten Reset wieder in den ausgangszustand zurückversetzt werden.
Um dies zu vermeiden, ist in der Mainloop ein Zähler implementiert, der bei jedem Schleifendurchlauf, in dem kein Datentransfer stattfindet und die Statemachine des GPS-Modus’ oder des PC-Modus’ nicht in ihrem Ausgangszustand ist, um 1 dekrementiert wird. Bei jedem Datenverkehr wird dieser Zähler wieder auf seinen Anfangszustand zurückgesetzt. Erreicht der Zähler den Wert 0, wird eine Timeoutfehlermeldung als Status ausgegeben, die aktuelle Session verworfen und die Statemachines in ihren ausgangszustand zurückversetzt.

## *5.3 Beschreibung grundlegender Datenstrukturen*

### 5.3.1 Verwendete Konstanten

Die wichtigsten Konstanten werden im Folgenden kurz beschrieben.

| Name der Konstanten | Kurze Beschreibung |
|---|---|
| `C_MemStart = 14;` | Anzahl Bytes des Headers im SRAM |
| `C_DLE = 16;`<br>`C_ETX = 3;` | Konstanten für Anfangs- und Endebytes eines Paketes |
| `C_Pid_Ack_Byte = 6;`<br>`C_Pid_Nak_Byte = 21;`<br>`C_Pid_Protocol_Array = 253;`<br>`C_Pid_Product_Rqst = 254;`<br>`C_Pid_Product_Data = 255;`<br>`C_Pid_Command_Data = 10;`<br>`C_Pid_Xfer_Cmplt = 12;`<br>`C_Pid_Records = 27;`<br>`C_Pid_Trk_Data = 34;`<br>`C_Pid_Wpt_Data = 35;`<br>`C_Pid_Trk_Hdr = 99;` | Konstanten für alle Packet IDs der im Programm vorkommenden Pakete |
| `C_Cmnd_Transfer_Wpt = 7;`<br>`C_Cmnd_Transfer_Trk = 6;` | Konstanten für die relevanten Kommandodaten eines Records Paketes |
| `C_Product_Data : array[0..44] of byte = (`<br>`016,255,039,141,000,240,000,101,`<br>`084,114,101,120,032,083,117,109,`<br>`109,105,116,032,083,111,102,116,`<br>`119,097,114,101,032,086,101,114,`<br>`115,105,111,110,032,050,046,052,`<br>`048,000,097,016,003);` | komplettes "Product_Data" Paket |
| `C_Protocol_Array : array[0..65] of byte = (`<br>`016,253,060,080,000,000,076,001,`<br>`000,065,010,000,065,100,000,068,`<br>`108,000,065,201,000,068,202,000,`<br>`068,108,000,068,210,000,065,045,`<br>`001,068,054,001,068,045,001,065,`<br>`244,001,068,245,001,065,088,002,`<br>`068,088,002,065,188,002,068,188,`<br>`002,065,032,003,068,032,003,219,`<br>`016,003 );` | komplettes "Protocol_Array" Paket |
| `s_ProductReq = 0;`<br>`e_ACKproductReq = 1;`<br>`e_ProductData = 2;`<br>`e_Protocol = 3;`<br>`s_Command = 4;`<br>`e_ACKcommand = 5;`<br>`e_Records = 6;`<br>`e_WptData = 7;`<br>`e_TrkData = 8;`<br>`e_Cmplt = 9;`<br>`NachCmplt = 10;` | Konstanten für die einzelnen Stati im PC-Mode Kommunikationsablauf |
| `e_ProductRequest = 0;`<br>`s_ProductData = 1;`<br>`e_ACKproductData = 2;`<br>`s_Protokollarray = 3;`<br>`e_ACKprotocolArray = 4;`<br>`e_CommandData = 5;`<br>`s_PidRecords = 6;` | Konstanten für die einzelnen Stati im GPS-Mode Kommunikationsablauf |

| | |
|---|---|
| `e_ACKrecords = 7;`<br>`x_setPointerBack = 8;`<br>`s_Points = 9;`<br>`e_ACKwptData = 10;`<br>`e_ACKtrkData = 11;`<br>`x_repeatComplt = 12;`<br>`e_ACKCmplt = 13;`<br>`i_Pointer = 14;` | |

### 4.3.2 Eigene Typen

Auflistung und Beschreibung aller eigenen Typen, die im Programm verwendet werden:

| Typdeklaration | Beschreibung |
|---|---|
| `TLinkStates = ( StartDLE, PacketID, PacketSize, EmpfangeDaten, Checksum, EndDLE, ETX );` | Dieser Typ definiert alle States der unteren Paketebene. Er dient dem korrekten Empfangen und Auswerten der einzelnen Bytes der Pakete. |
| `TRichtung = ( Hoch, Runter, Nix );` | Variablen dieses Typs geben an, ob von der aktuellen Speicheradresse weiter hochgezählt (Hoch) oder weiter runtergezählt (Runter) wird. Alternativ bleibt die aktuelle Adresse unverändert, wenn „Nix“ angegeben wird. |

### 4.3.3 Globale Variablen

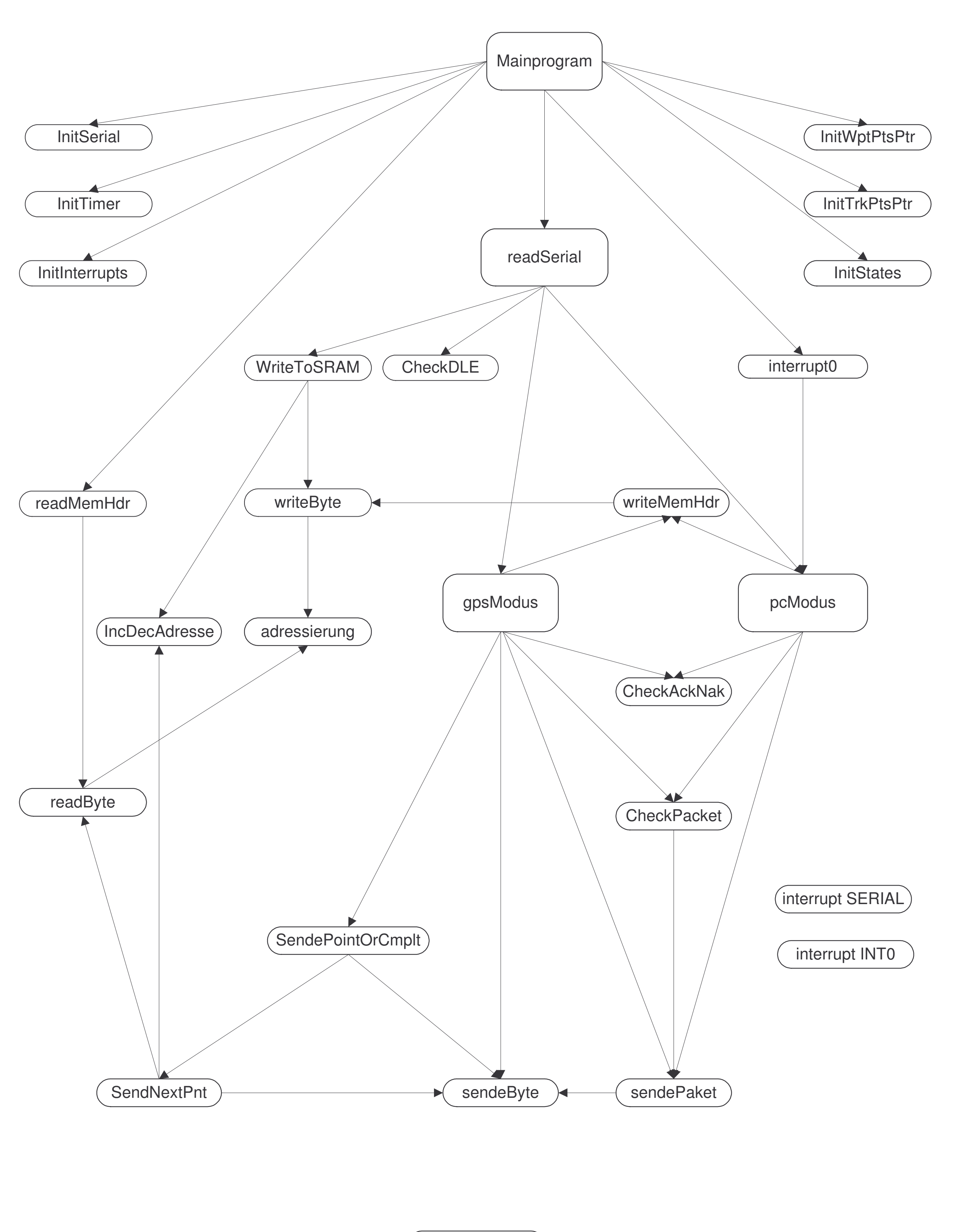
Bei der Programmierung des Atmel-Mikroprozessors mit dem Pascal System51 System stehen dem Programmierer 128 Byte Arbeitsspeicher zur Verfügung. Da sowohl Funktionsparameter als auch funktionslokale Variablen diesen Speicherbereich nutzen müssen, wurden viele Variablen global deklariert, um den Speicherplatz durch Mehrfachnutzung besser ausschöpfen zu können, auch wenn dies auf Kosten der Lesbarkeit des Quellcodes geht.

Auflistung und Beschreibung aller globalen Variablen, die im Programm verwendet werden:

| Typdeklaration | Beschreibung |
|---|---|
| `port1 : byte at P1;`<br>`isLesen : boolean at P3.7;`<br><br>`isMemActive : boolean at P3.5;`<br><br>`adr3 : boolean at P3.3;`<br>`adr4 : boolean at P3.4;`<br><br>`isStatus : boolean at P3.2;` | - Variable port1 als Bytewert an Port P1 deklarieren<br>- Ausgang zum Setzen des SRAM-Zugriffsmodus: lesen = true, schreiben = false<br>- Ausgang zum aktivieren des SRAMs: standby = false, Aktiv = true }<br>- Ausgänge zur Steuerung der Clock-Signale der Adress-Flipflops<br><br><br>- Ausgang zur Steuerung des Clock-Signals des Status-Flopflops |
| `fl_readSerial : boolean;`<br><br>`fl_int0 : boolean;` | - Flag zur Überprüfung, ob ein Interrupt der Seriellen Schnittstelle aufgetreten ist<br>- Flag zur Überprüfung, ob ein Interrupt am |

| | |
|---|---|
| | Externen Eingang 0 (Taster) aufgetreten ist |
| fl_senden : boolean; | - Flag zur Überprüfung, ob ein Paket gesendet werden soll |
| fl_memfull : boolean; | - Flag zur Speicherüberlaufsprüfung |
| fl_wakeup : boolean; | - Dieses Flag wird gesetzt, wenn der Atmel aus dem Standby Modus „erwacht“ |
| TIsendet : boolean; | - Zeigt an, ob gerade Daten über die Serielle Schnittstelle gesendet werden |
| g_DLEflag : boolean; | - Flag zur Überprüfung, ob ein DLE-Byte in den Bytes eines Paketes empfangen worden ist |
| g_PacketID : byte; | - speichert die ID des aktuell empfangenen Paketes |
| linkstate : TlinkStates; | - speichert den aktuellen Zustand beim Empfangen eines Paketes auf Byte-Ebene |
| g_sizeofpacket : byte; | - speichert das Längenbyte des gerade empfangenen Pakets |
| checksumcounter : byte; | - aktualisiert die Checksumme nach jedem empfangenen Datenbyte |
| dummy : byte; | - Verschiedenes |
| g_Richtung : TRichtung; | - gibt an, ob und wo im SRAM das aktuell empfangene Paket abgelegt werden soll |
| g_ChecksumOkay : boolean; | - speichert, ob die Checksumme des zuletzt empfangenen Pakets richtig war |
| g_DataByte0 : byte; | - speichert das erste Datenbyte das aktuell empfangenen Pakets |
| g_DataByte1 : byte; | - speichert das zweite Datenbyte das aktuell empfangenen Pakets |
| g_numBytesEmpfangen : byte; | - speichert die Anzahl der schon empfangenden Datenbytes eines Pakets |
| g_TrkPtsPtrHigh : byte;<br>g_TrkPtsPtrMid : byte;<br>g_TrkPtsPtrLow : byte; | - zeigt immer auf die Adresse, in der das aktuell zu empfangene Byte geschrieben bzw. aus der das aktuell zu sendende Byte gelesen werden soll |
| g_WayPtsPtrHigh : byte;<br>g_WayPtsPtrMid : byte;<br>g_WayPtsPtrLow : byte; | - zeigt immer auf das erste freie Byte hinter den Waypoints der letzten Session |
| g_TmpPtsPtrHigh : byte;<br>g_TmpPtsPtrMid : byte;<br>g_TmpPtsPtrLow : byte; | - zeigt immer auf das erste freie Byte hinter (vor) den Trackpoints der letzten Session |
| g_TmpStartPtrHigh : byte;<br>g_TmpStartPtrMid : byte;<br>g_TmpStartPtrLow : byte; | - zeigt immer auf das erste Byte nach dem Ende des zuletzt komplett empfangenen Pakets |
| g_TmpPaketPtrHigh : byte;<br>g_TmpPaketPtrMid : byte;<br>g_TmpPaketPtrLow : byte; | - zeigt immer auf den Anfang eines Pakets (auf Paketebene), um ggf. bei defekten Paketen wieder an die alte Speicheradresse zurückspringen zu können |
| g_isPC : boolean; | - bestimmt den aktuellen Arbeitsmodus: "Gerät ist PC" oder "Gerät ist GPS" |
| g_PcState : byte; | - speichert den aktuellen Zustand beim Empfangen einer Session auf Paket-Ebene im PC-Modus |
| g_GpsState : byte; | - speichert den aktuellen Zustand beim Senden aller Daten auf Paket-Ebene im GPS-Modus |
| g_Packetcounter : word; | - gibt an, wie viele Pakete noch empfangen bzw. noch gesendet werden müssen |
| g_PcWillWayPts : boolean; | - gibt an, ob die aktuelle Session WayPts (=true) oder TrkPts (=false) behandelt |
| g_numPakete : word; | - speichert, wie viele Pakete in akt. Session empfangen werden sollen (Pid_Records) |
| g_numPaketeTrk : word; | - speichert, wieviele TrkPt-Pakete empfangen wurden und im SRAM liegen (für Pid_Records) |
| g_numPaketeWpt : word; | - speichert, wieviele WayPt-Pakete empfangen wurden und im SRAM liegen (für Pid_Records) |
| g_timer : word; | - dient zur Abfrage von Timeouts |
| g_timerhelp : byte; | - dient zur Abfrage von Timeouts |

## *5.4 Programmorganisationsplan*



## 5.5 Programmtests

Die wichtigsten kritischen Programmabläufe wurden getestet und zusammengestellt:

| Testfall | Erwartetes Ergebnis | Erzieltes Ergebnis |
|---|---|---|
| Serielles Kabel wird während einer Übertragung durchtrennt. | Die Schaltung sollte Timeout-Fehler auf der 7-Seg-Anzeige melden und die Übertragung abbrechen. | Schaltung meldet Timeout-Fehler auf der 7-Seg-Anzeige und bricht Übertragung ab. |
| Während einer Übertragung wurde ein Paket durch ein Nak-Paket erneut angefordert. | Die Schaltung sollte das entsprechende vorige Paket erneut senden. | Die Schaltung verschickt das vorige Paket erneut. |
| Während einer Übertragung wird ein Paket mit ungültiger Packet-ID empfangen. | Die Schaltung sollte das Paket ignorieren, kein ACK-Paket zurückschicken und auf ein gültiges Paket warten. | Die Schaltung ignoriert das Paket, sendet kein ACK-Paket zurück, sondern wartet auf ein gültiges Paket. |
| Während einer Übertragung wird ein Paket mit ungültiger Checksumme empfangen. | Die Schaltung sollte den Fehler erkennen und ein Nak-Paket mit Referenz auf das fehlerhafte Paket zurückschicken. | Die Schaltung sendet ein Nak-Paket mit Referenz auf das fehlerhafte Paket zurück. |
| Während einer Übertragung wird ein gültiges Paket eines anderen States empfangen. | Die Schaltung sollte das Paket ignorieren, kein ACK-Paket zurückschicken und auf ein gültiges Paket warten. | Die Schaltung ignoriert das Paket, sendet kein ACK-Paket zurück, sondern wartet auf ein gültiges Paket. |
| Inmitten einer Übertragung wird der PC/GPS-Modus-Taster gedrückt. | Der Taster sollte während einer Übertragung deaktiviert sein und keinen Interrupt auslösen. | Der Taster löst keinen Interrupt aus, sondern bleibt ohne Funktion. |
| Bei der Übertragung von Waypoints werden nur drei Way-Points empfangen, obwohl in Pid_Records eine höhere Zahl stand. | Die Schaltung sollte auf das fehlende Waypoint-Paket warten. Kann keins empfangen werden, muss der Timeout ausgelöst werden. | Die Schaltung wartet auf das fehlende Waypoint-Paket. Nach ca. 2 Sekunden kommt ein Timeout. |
| Im GPS-Modus werden Waypoints angefragt und unsere Schaltung ist noch leer. | Die Schaltung sollte einen gültigen Durchlauf starten, mit PidRecords=0 und keinen folgenden Waypoints. | Die Schaltung beantwortet die Anfrage und schickt ein Paket mit PidRecords=0 und keine folgenden Waypoints los. |

# 6. Anhang

## *6.1 CD-ROM mit Quellcode und Dokumentation (PDF)*

## *6.2 Pascal-Quellcode des Programms für den Atmel-Prozessor*